Application du Singleton

Pour tous les exercices de ce TD, chaque manipulation de chaînes de caractères, de choix d'un conteneur pourra se faire à travers l'utilisation du framework de la stl.

Ce TD a pour but la réalisation d'une application permettant de simuler le trafic maritime dans un port.

Exercice 1

Créer une classe **Ship** contenant un attribut qui permet de définir un nom. A travers cette classe il sera possible d'accéder à cette valeur et de la modifier. (Penser aux accesseurs). *Le nom des bateaux sera représenté sous une forme de chaînes de caractères à l'aide de la classe std::string ou QString.*

Exercice 2

L'exercice 2 permet de définir un objet Harbor représentant le port dans lequel les bateaux seront déplacés. C'est l'équivalent d'une surface plane navigable (en 2D). Il sera possible de déplacer les bateaux en modifiant leur position.

1°) Créer une classe ***Harbor*** sachant que :

- un port est une surface navigable plane correspondant à une matrice[N][M].
- chaque position est définie par des coordonnées (x,y).
- Chacune des positions peut contenir à un instant *t* soit un vaisseau (Ship*) soit être vide (NULL).

La déclaration de la matrice pourra se faire à l'aide du conteneur std ::map ou QMap.

2°) la classe ***Harbor*** doit contenir une méthode capable de déplacer une instance de *Ship* d'un point de départ à un point d'arrivé sachant que :

- Si la position d'arrivée est déjà occupée, l'objet ***Harbor*** ne modifie pas la position du bateau et indique que la place est déjà prise.
- Si la place est libre, l'objet ***Harbor*** affecte l'instance à cette position et libère celle de départ.

3°) Le classe ***Harbor*** doit être également capable de définir des quais auxquels les bateaux pourront venir s'amarrer. La notion de quai est représenté par un identifiant et un point sur la 1ère ou la dernière colonne de la matrice, comme ci-dessous :

1									12
4									20
2									29
15									11
16									28
5									19
21									27
22									10
6									18
7									17

Les identifiants peuvent être aléatoires et n'ont aucun rapport avec leur position dans la matrice.

Modifier la classe ***Harbor*** pour que celle-ci définisse la liste des quais disponibles en fonction de la taille de la matrice. Une fois les quais définis, la classe doit fournir une méthode permettant d'obtenir les coordonnées d'un quai par rapport à son identifiant.

4°) Définir une méthode dans la classe ***Harbor*** qui permet de :

- réserver un numéro de quai avec le nom d'une instance de *Ship*. *Il n'est pas possible de réserver un quai déjà affecté.*
- donner le numéro d'un quai réservé en fonction du nom d'un bateau. *Si le bateau n'a pas de quai affecté, la méthode retourne -1 ou 0.*

5°) Définir une méthode *getInitialPositions*() dans la classe ***Harbor*** qui retourne la liste des positions d'entrée dans la matrice (en vert sur la figure). *Les nouvelles instances de la classe Ship ne pourront être affectées à la matrice qu'à travers ces coordonnées d'entrée.*

(Optionnel) 6°) A chaque appel, les méthodes implémentées ci-dessus devront écrire dans un fichier unique (*par exemple "trafic.txt"*) les informations suivantes :

- l'heure de l'appel de la fonction, précédée de l'instruction TIME:
- L'ajout d'un nouvel objet *Ship* dans l'instance de classe ***Harbor,*** précédé de l'instruction ADD:
- le nom de l'objet *Ship* ainsi que sa position à chaque déplacement, précédé de l'instruction MOVE :
- L'impossibilité de déplacer un objet si la place est occupée, précédé de l'instruction ERR
- Le numéro de quai lors d'une affectation, précédé de l'instruction BIND

Un exemple de fichier :

TIME: 10:45:25
ADD: ship1
TIME: 10:45:28
BIND: ship1 15
TIME: 10:45:30
MOVE: ship1, (8,5) (8,6)
…

7°) Enfin, l'objet ***Harbor*** sera mené à gérer le déplacement de plusieurs bateaux à travers des threads différents. Transformer donc la classe ***Harbor*** en Singleton.

Exercice 3 : (modification de l'exercice)

L'objectif de cet exercice est d'établir une cohésion entre les différents exercices du projet, d'obtenir un code dynamique à partir de la définition statique des classes précédemment créées. Votre code devra générer des instances de la classe Ship et être en charge des déplacements dans la matrice.

1°) Créer une classe dans laquelle sera définie une méthode nommée *cycle.*

2°) Définir dans cette même classe une méthode qui doit pouvoir générer une instance de la classe *Ship*. (dont le prototype sera par exemple *Ship* createShip()*).

3°) lorsque la méthode *cycle* est appelée, elle devra réaliser les instructions suivantes :

1. génération d'une instance de la classe *Ship* avec une probabilité *Pg*.
2. a. Affectation de l'instance si possible à l'une des deux entrées de libre dans la matrice
   b. Sinon, l'instance doit être empilée dans une file d'attente.
3. Réservation d'un quai libre à l'instance nouvellement créée.
4. Calcul du vecteur trajectoire vers les coordonnées du quai pour chaque instance dans contenu dans la matrice.
5. Déplacement de chaque instance d'un point en direction de la position du quai tant que la position d'arrivée n'est pas atteinte. (Appel à la méthode qui convient de la classe ***Harbor*** définie dans l'exercice 2)
6. Pause de 1 seconde et retour au point 1.

Encapsuler lorsque cela semble nécessaire les étapes du comportement dans des méthodes différentes pour plus de lisibilité.

N.B : faites varier le paramètre Pg *et observer le comportement.*

Rappels et outils :

Quelques algorithmes simples de recherche et d'ajout dans les conteneurs de la stl…

Classe	Un ajout	une recherché

std::map	`map<string, int> myMap ;`
	`string key ;`
	`int value ;`
	`myMap[key] = value ;`
	
	ou
	
	`myMap.insert(std::make_pair(key,value));`
		`if (myMap.find(key) != myMap.end() )`
		`{`
		`  return myMap[key];//retourne value.`
		`}`
		`else`
		`{`
		`  //il n'existe aucune valeur à l'index 'key'`
		`  return -1 ;`
		`}`

QMap	Idem	`if (myMap.contains(key) )`
		`{`
		`  return myMap[key]; //retourne value.`
		`}`
		`else`
		`  return -1 ;`

std::vector	`vector<string> myVector;`
	`myVector.push_back("toto");`
	`myVector.push_back("tutu");`
	`myVector.push_back("titi");`
		`int i = 2;`
		`if (i < myVector.size() )`
		`  return myVector[i]; //retourne "titi"`

Les iterations:

std::map	std::vector

`map<string, int> myMap;`
`map<string, int>::iterator it = myMap.begin();`
 	
`while (it != myMap.end())`
`{`
`  it->first; //< j'accède à la clef de type string.`
`  it->second; //< j'accède à la valeur de type int.`
`  it++;`
`}`
	`vector<int> myvector;`
	`for (int i=1; i<=5; i++)`
	`  myvector.push_back(i);`
	
	`vector<int>::iterator it;`
	
	`cout << "myvector contains:";`
	
	`for ( it=myvector.begin() ; it < myvector.end(); it++ )`
	`  cout << " " << *it;`

Aide pour les coordonnées :

Pour les coordonnées à définir dans la classe **Harbor** il est possible d'utiliser la classe `std::pair<int,int>` (ou `Qpair<int,int>`). Mais cette classe ne pourra pas être utilisée en guise de clef pour l'indexation des objets dans la matrice (i.e : quelque chose comme ceci : `map<pair<int,int>, MyClass*>`) car elle ne contient pas l'opérateur **<**. Dans l'implémentation de ce cas, il est fortement conseillé de définir une nouvelle classe (ou structure) coordonnées, comme suit :

```
class coord
{
   public :
   int x ;
   int y ;
   bool operator < (const coord& c) ;//< A définir !!
   …
} ;
```

Puis,

```
map<coord, MyClass*>
```